Spring Integration and EIP

TIKAL

Agenda

- Introduction to Spring Integration
- Enterprise Integration Patterns
- Demo
- Spring Integration Compared
- Summary and questions

TIKAL

The synchronous breakdown

- A customer orders coffee
  - » and waits ...
- A waiter walks to barista and passes the order
  - » and waits ...
- A barista walks to the coffee machine
  - » and waits ...
- How about the next customer?

What is messaging?

- Multiple agents should work together
- Without being in each other way
- Waiter helps customer and cook to collaborate

Characteristics of messaging

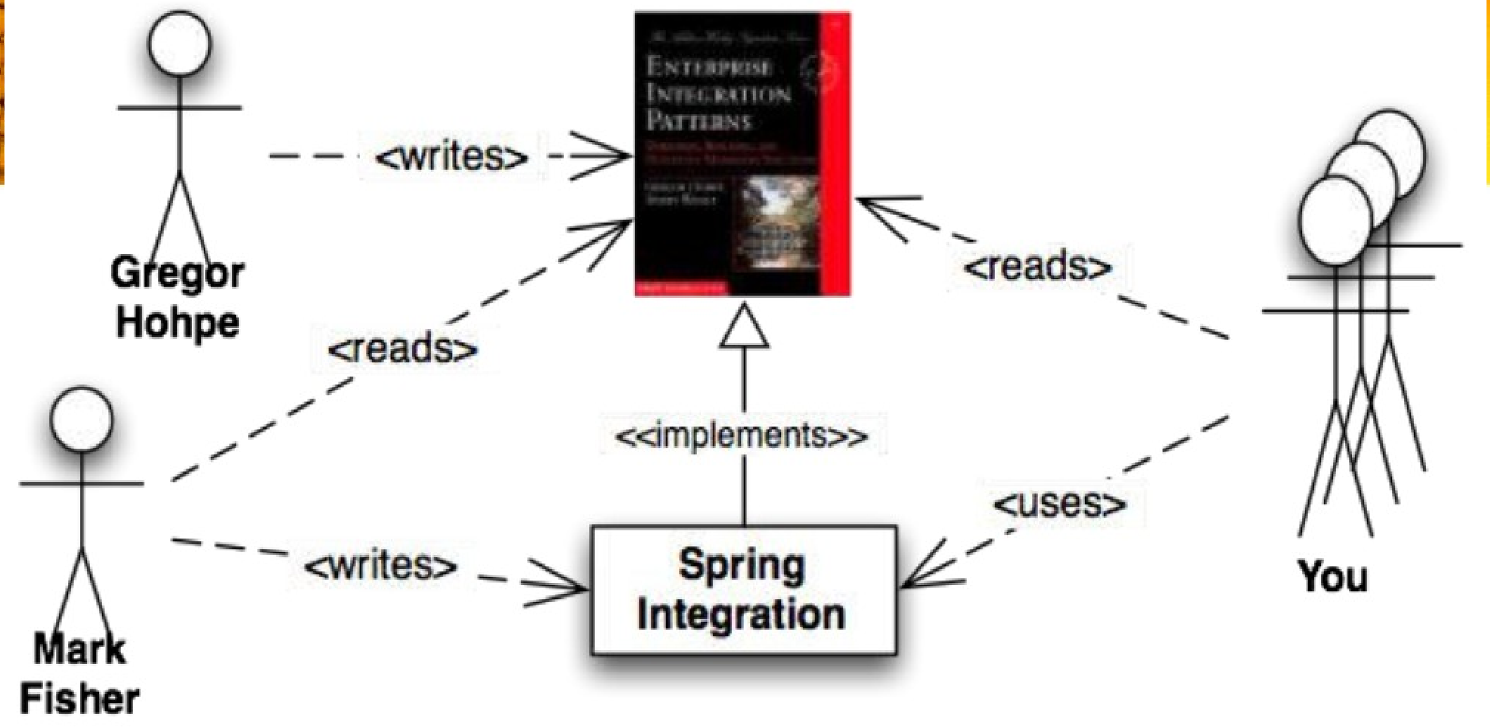
- Transport
  - The waiter takes an order and moves it to barista
- Asynchronous
  - The waiter and the barista work on other orders instead of waiting for every order to complete
- Translation
  - Waiter uses different terms with customer and the barista
- Routing
  - Orders arrive back at the proper table

TIKAL

Why messaging?

- Loose coupling
- Performance
  - Much higher throughput
- Flexibility
  - Waiters can replace each other
- Interception and filtering
  - A waiter can replace a dirty mug before the customer notices

Spring Integration



TIKAL

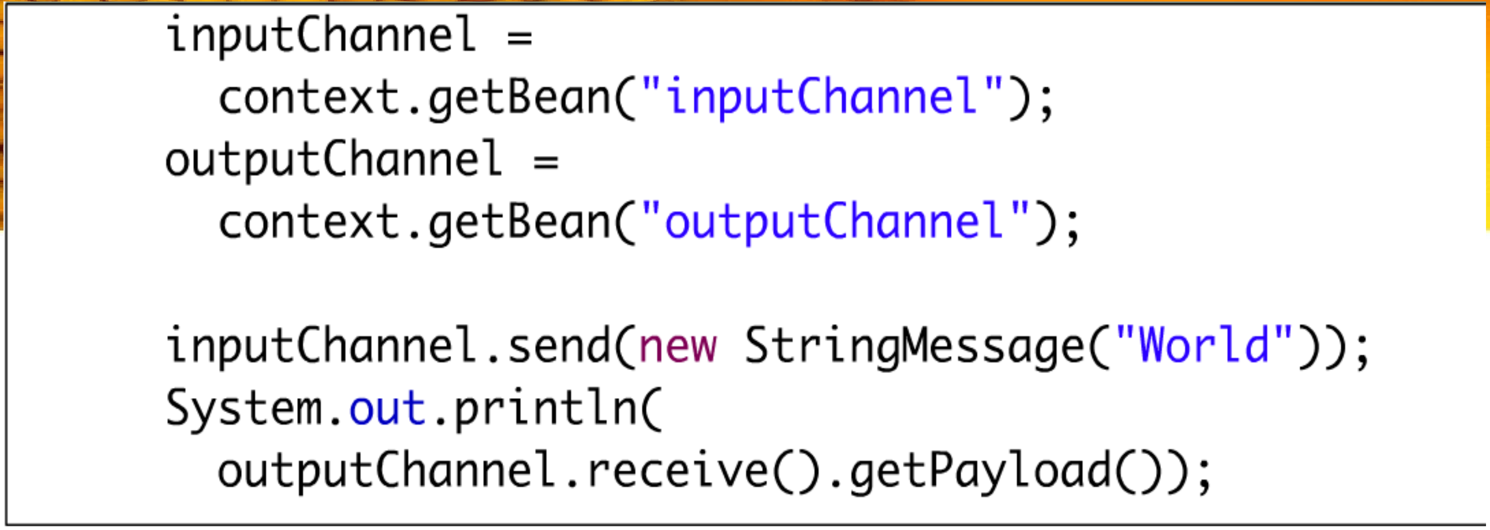
Hello world (XML)

```
<service-activator input-channel="inputChannel"
          default-output-channel="outputChannel"
          ref="helloService"
          method="sayHello"/>

<beans:bean id="helloService" class="...HelloService"/>
```

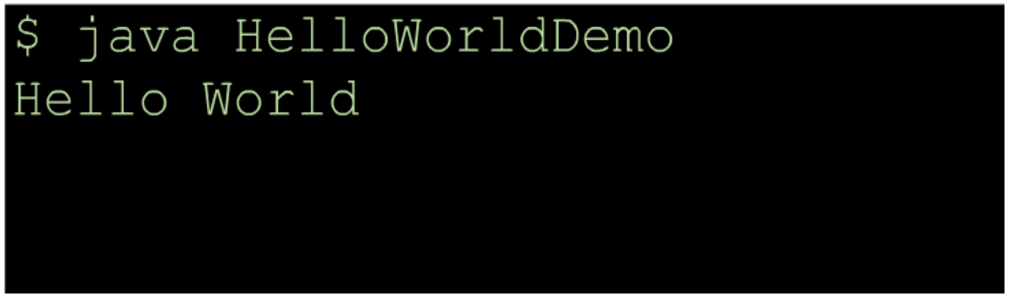
```
public class HelloService {
  public String sayHello(String name) {
    return "Hello " + name;
  }
}
```

Hello world (Java)



```java
inputChannel =
  context.getBean("inputChannel");
outputChannel =
  context.getBean("outputChannel");

inputChannel.send(new StringMessage("World"));
System.out.println(
  outputChannel.receive().getPayload());
```

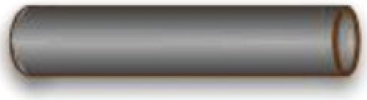


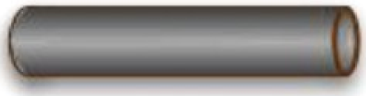
```
$ java HelloWorldDemo
Hello World
```

Channels

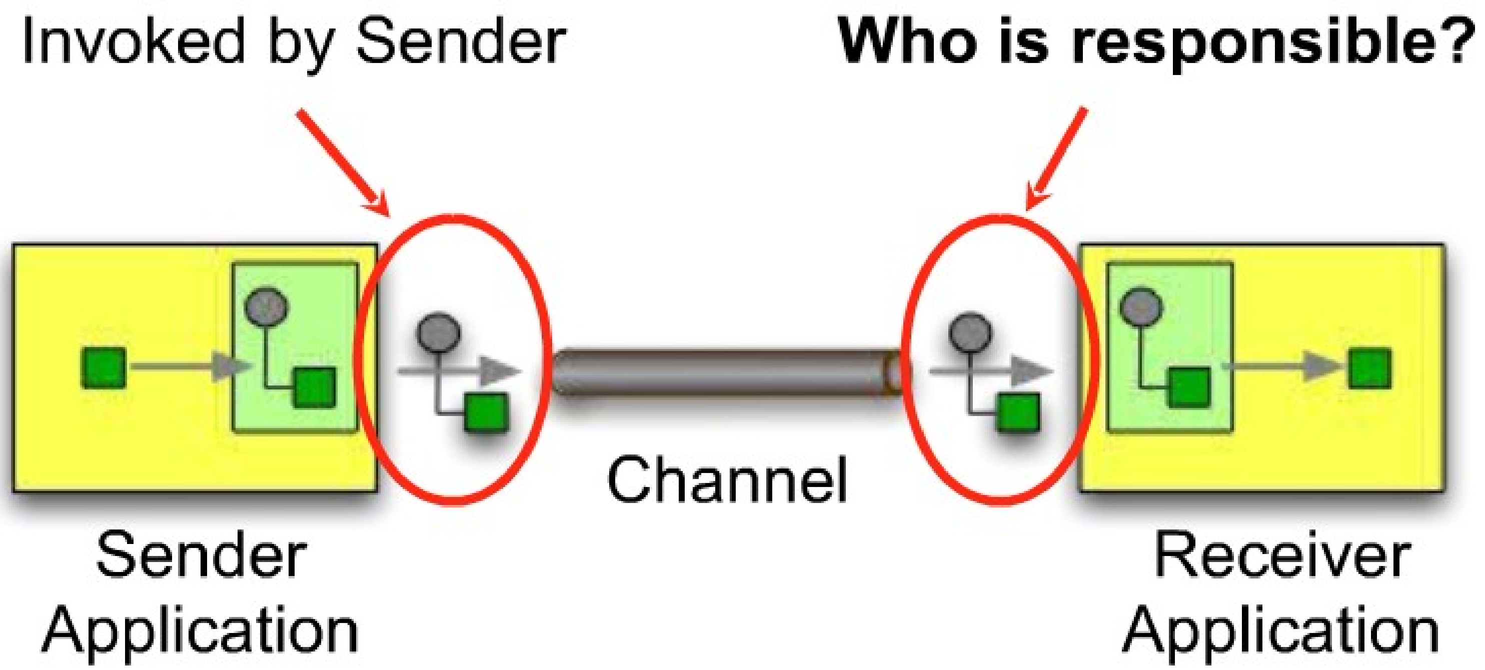
```
<channel id="incoming"/>

<channel id="orderedNotifications">
    <queue capacity="10"/>
</channel>
```

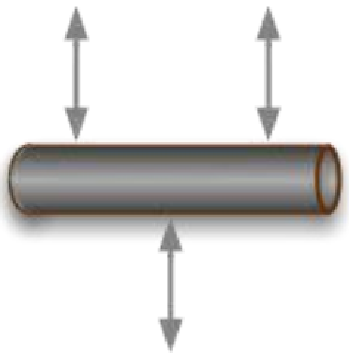




Message Endpoints



The poller



A Message Bus enables separate applications to work together, but in a decoupled fashion such that applications can be easily added or removed without affecting the others.

```
<poller default="true"/>
```

In Spring Integration:

you don't need to worry about it

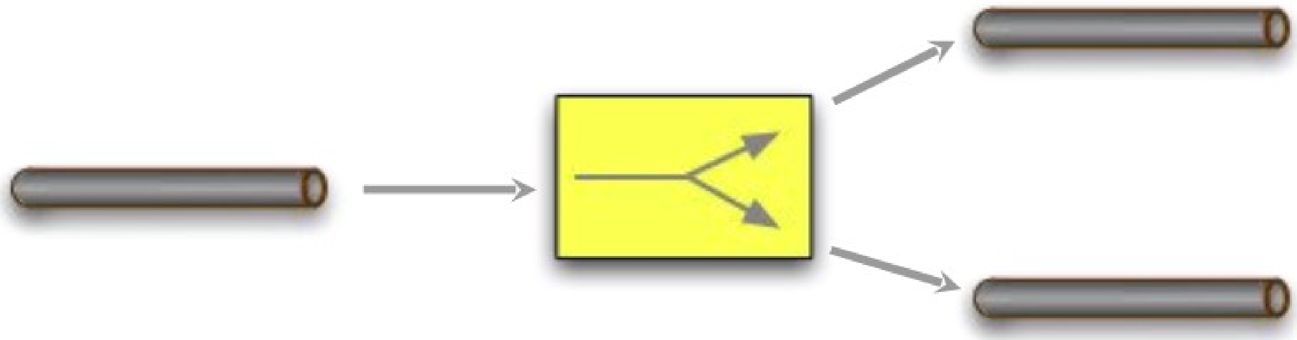
Spring Integration

- Introduction to Spring Integration
- **Enterprise Integration Patterns**
- Demo
- Spring Integration Compared
- Summary and questions

TIKAL

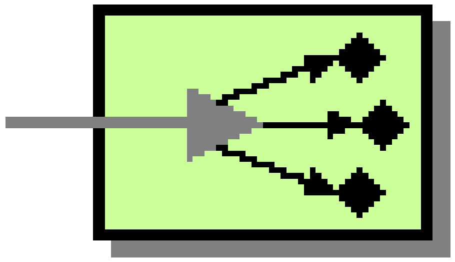
Message router

- Takes messages from a channel and forwards them to different channels



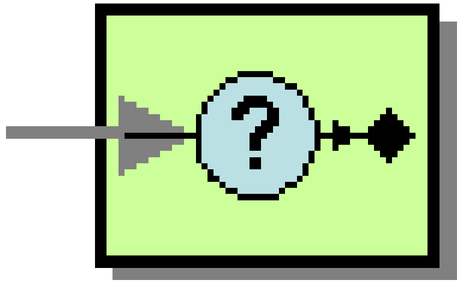
Competing consumers

- Multiple consumers take messages from the channel
- First come, first serve



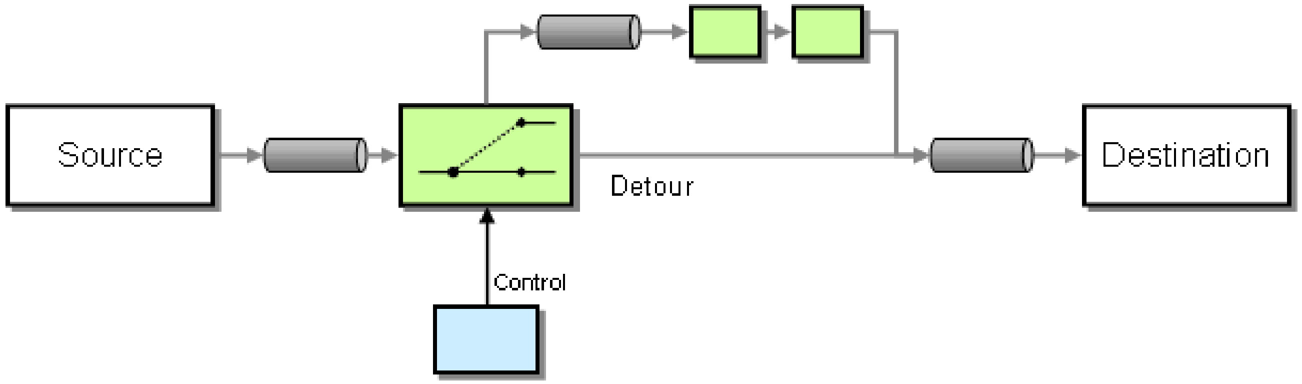
Selective consumer

- Select only relevant messages
- Reduces the need for dedicated channels



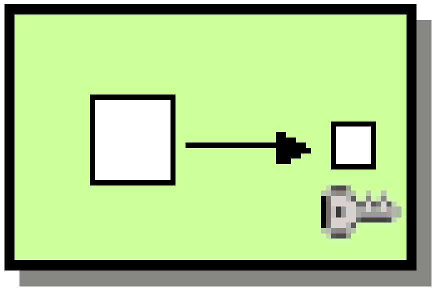
Detour

- Allows to send messages through additional steps if a control condition is met.
- Useful for turning on extra validation, test, debugging, etc.



Claim check

- Send the information across the system without sacrificing its content



Delayer

- Ensure message waits for a certain amount of time before being delivered
- Now Spring Integration has an effective implementation of this pattern

TIKAL

Demo

Spring Integration

- Introduction to Spring Integration
- Enterprise Integration Patterns
- Demo
- **Spring Integration Compared**
- Summary and questions

TIKAL

What's different about Spring Integration?

- Can be used from within an existing application
- Lightweight (like any Spring application)
  - run from JUnit test
  - run within web application
- Focused on messaging and integration
- Not an ESB

Others in general

- Full blown ESB
- It's an application, not a framework
  - You need to install it
  - You need to run it
- Typically a lot heavier
- Focus on the deployment architecture, not the actual integration

Mule

- Integrates with Spring
- Lots of integration options
  - SOAP, REST
  - JMS
- Embeddable
- Distribution
  - 32 MB
  - 2 MB jar only (for embedding)

ServiceMix

- Designed as JBI implementation
  - Straight-through, SEDA, JMS and JCA flows
  - all messages only in XML
  - much slower than direct calls
- Distribution 100 MB
- Excellent XML configuration and routing

TIKAL

Open ESB

- JBI based
- Requires GlassFish server
  - > 100 MB distribution
- WSDL used for all types of endpoints
  - WSDL for email messages
  - WSDL for database tables
- BPEL used for all processing
- Excellent GUI for data transformation and BPEL

TIKAL

Camel

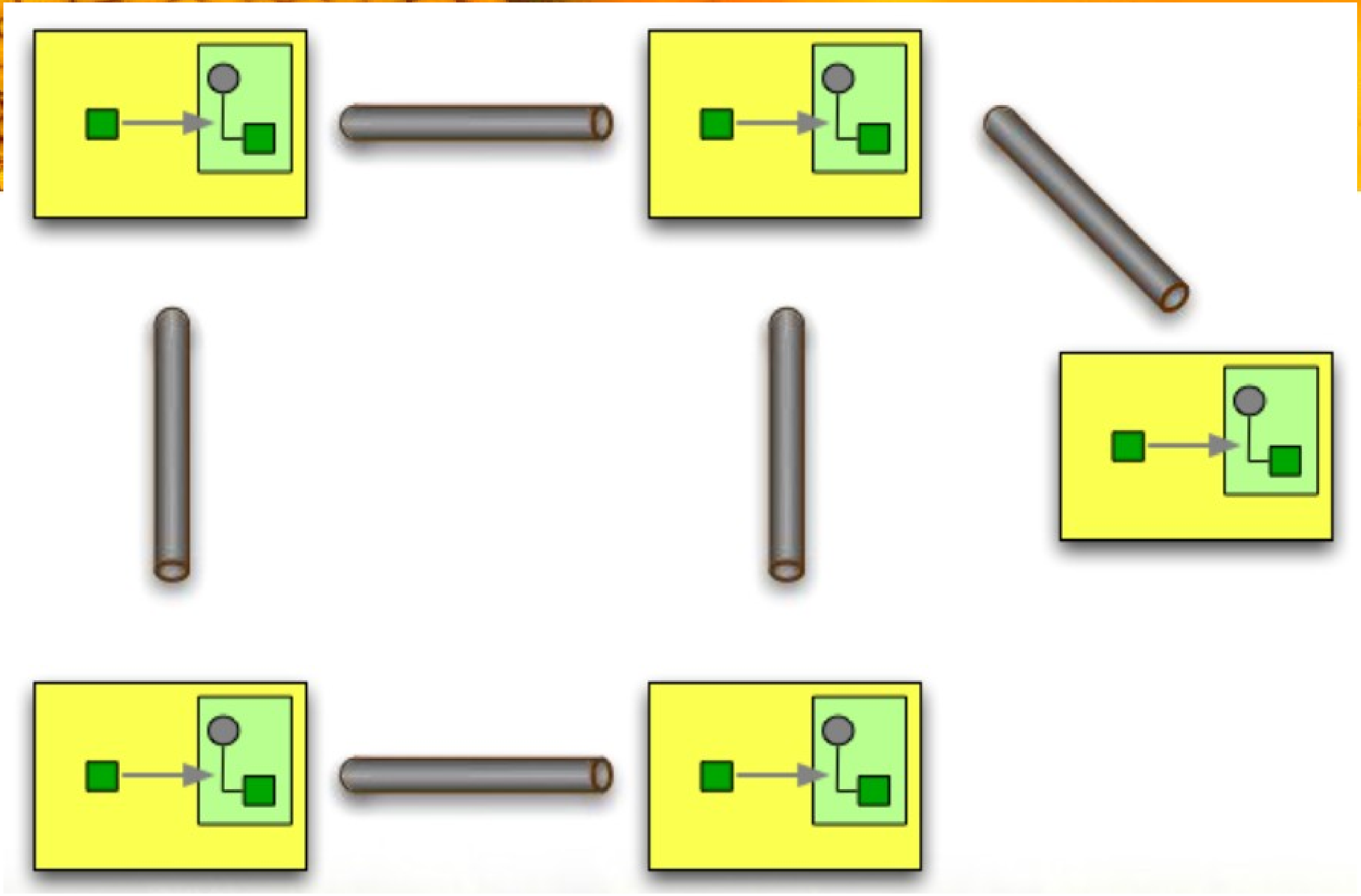
- Most direct Camel competitor
- Lest consistent with Spring
- Focus on routing
- Fluent API as an alternative to XML

Considerations

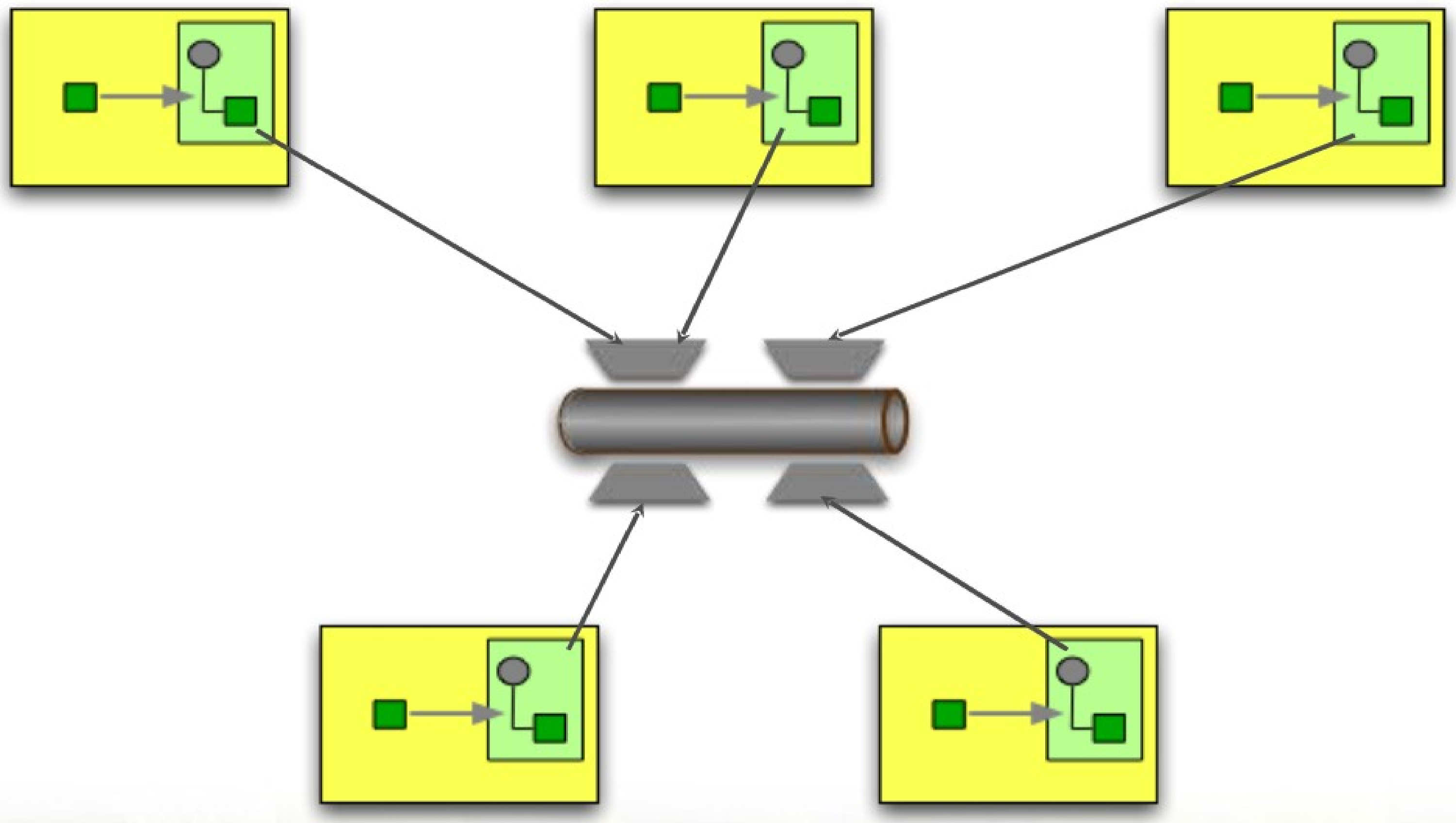
- Routing complexity
  - Bus can be used for complex routing problems
- Coarse grained
  - less channels, less universal endpoints
- Fine grained
  - more channels, small reusable endpoints
- Keep it clean
  - multiple configuration files can be used to keep it manageble

TIKAL

Integration without a bus



Integration with a bus



References

- This presentation heavily uses
  http://www.slideshare.net/iweinfuld/spring-integration-and-eip-introduction
- Spring Integration home
  http://www.springsource.org/spring-integration
- Enterprise Integration Patterns list
  http://camel.apache.org/enterprise-integration-patterns.html

Spring Integration

- Introduction to Spring Integration
- Enterprise Integration Patterns
- Demo
- Spring Integration Compared
- **Summary and questions**

TIKAL

Summary

- Spring Integration
  - Works from existing Spring application
  - Lightweight
  - Decentralized (if you want)
- Enterprise Integration Patterns
  - describe ways of plumbing of loose coupled and (possibly) asynchronous components
  - plays same role as design patterns for traditional applications

TIKAL

Q & A

Thank you